Journal of The Korea Society of Computer and Information
Vol. 19, No. 10, October 2014
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2014.19.10.081

SA 해쉬 알고리즘을 이용한 중복파일 업로드 방지 시스템 설계

황 성 민*, 김 석 규*

Design of System for Avoiding upload of Identical-file using SA Hash Algorithm

Sung-Min Hwang*, Seog-Gyu Kim*

요 약

본 논문에서는 서버로의 중복파일을 업로드 방지를 위하여 SA 해쉬 알고리즘을 제안하고 이를 이용하여 서버 시스템을 설계한다. SA 해쉬 값으로 동일한 파일이 서버에 있는 지 검사하고 존재한다면 클라이언트에게 업로드를 받지 않고 기존 파일을 이용하는 방법으로 효율적인 시스템 설계를 할 수 있게 되는 것이다.

중복파일 검사를 할 수 있는 SA 해쉬 알고리즘은 출력하고자 하는 비트 수 n을 한 블록으로 하고 원본 파일을 블록 단위로 나누게 된다. 원본 파일의 mod i 비트와 출력 해쉬 값의 i 비트를 XOR 연산을 하게 된다. 이렇게 반복적으로 원본 파일 길이까지 XOR연산을 하는 것이 SA 해쉬 알고리즘의 메인 루틴이다. 기존 해쉬 함수인 MD5, SHA-1, SHA-2보다 중복파일 업로드 방지 서버 시스템에 적합한 해쉬 함수인 SA 해쉬 알고리즘을 통해 시간 및 서버 스토리지 용량의 절약을 도모할 수 있다.

▸ Keywords : 중복파일, 해쉬 알고리즘, 대용량 서버

Abstract

In this paper, we propose SA hash algorithm to avoid upload identical files and design server system using proposed SA hash algorithm. Client to want to upload file examines the value of SA hash and if the same file is found in server system client use the existing file without upload.

SA hash algorithm which is able to examine the identical-file divides original file into blocks of n bits. Original file's mod i bit and output hash value's i bit is calculated with XOR operation. It is SA hash algorithm's main routine to repeat the calculation with XOR until the end of original file. Using SA hash algorithm which is more efficient than MD5, SHA-1 and SHA-2, we can design server system to avoid

• 제1저자 : 황성민 •교신저자 : 김석규
• 투고일 : 2014. 9. 17, 심사일 : 2014. 9. 25, 게재확정일 : 2014. 9. 30.
* 안동대학교 공과대학 정보통신공학과 (Dept. of Infomation Communication, College of Engineering, Andong National University)

uploading identical file and save storage capacity and upload-time of server system.

▸ Keywords : Identical-file, Hash algorithm, Server system

I. 서 론

하드웨어 성능이 빠른 속도로 발전을 하고 있지만 저장 데이터 및 파일의 크기 또한 늘어나고 있다. 그리고 최근 웹하드 및 클라우드 서비스가 등장하면서 파일 서버의 저장공간의 사용량이 많아졌다. 그러한 이유로, 하드웨어의 발전이 있더라도 자원 관리, 자원 절약은 항상 이슈가 되고 있다. 일반적으로 서버 시스템은 클라이언트로부터 업로드 요청을 받으면 서버에 동일한 파일이 존재하여도 또 다시 업로드를 받고, 그 파일을 서버에 보관하게 된다. 그럼 동일한 파일들이 서버의 스토리지 용량을 불필요하게 점유하게 되고 클라이언트에서 업로드 할 때 발생하는 트랙픽 증대 및 업로드 시간이 많아지고 있다.

본 논문에서는 SA 알고리즘을 제안하고 이를 이용한 시스템을 설계함으로써 클라이언트가 업로드 하고자 하는 파일이 서버에 동일한 파일이 있는 경우 업로드를 하지 않으므로 저장공간 절약과 트래픽 감소 및 업로드 시간을 줄일 수 있는 방법을 제시한다. 또한 서버에 있는 파일이 저작권 및 관리상 문제로 제거해야 되는 경우에도 쉽게 제거할 수 있는 시스템을 설계 및 구현이 가능하다.

본 논문에서는 해쉬 함수의 성능 분석과 본 논문에서 제안하는 시스템 설계에 가장 적합한 자체적으로 고안한 SA(Simple Arithmetic) 해쉬 알고리즘을 제안하고 있다. 2장에서는 기존 해쉬 알고리즘을 및 기존 서버 설계 방식에 관해 논하고 3장에서는 제안하는 SA 해쉬 알고리즘을 소개하며 SA 해쉬 알고리즘을 이용한 중복 파일 업로드 방지 서버 지스템의 설계 방식을 제안하고 있다. 4장에서는 기존 방식의 파일 서버 시스템과 기존 해쉬 알고리즘으로 파일 서버를 시스템을 설계하는 경우 그리고 본 논문에서 제안하는 SA 해쉬 알고리즘으로 중복파일 업로드 방지 시스템을 설계하는 경우의 성능 분석을 하게 된다. 5장에서는 실험을 기반 하는 결론을 도출 하고자 한다.

II.기존 Hash Algorithm 및 서버 형식

1. MD5 Hash algorithm

MD5는 임의의 길이의 메시지를 입력받아 128비트짜리 고정 길이의 해쉬 값을 출력한다 [1][2]. 입력 메시지는 512비트 블록들로 나누고, 나누어떨어지지 않는 파일의 마지막 부분은 패딩을 더하여 512비트로 만들어 준다. 블록의 첫 단일 비트 1을 블록 메시지 끝 부분에 추가하고 512의 배수의 길이 보다 64비트가 적은 곳까지 0으로 채운다. 나머지 64비트는 처음 메시지의 길이를 나타내는 64비트 정수 값으로 채워진다. 메인 MD5 알고리즘은 A,B,C,D 라고 이름 붙은 32비트 워드 네 개로 구성하고 그것을 128비트 state에 의해 동작시킨다. A,B,C,D는 소정의 상수값으로 초기화된다. MD5 알고리즘은 각각의 512비트짜리 입력 메시지 블록에 대해 차례로 동작하며, 512비트 입력 메시지 블록을 처리하고 나면 128비트 state의 값이 변하게 된다.

하나의 메시지 블록을 처리하는 것은 4단계로 나누어 처리한다. 한 단계를 라운드(round)라고 부르며, 각 라운드는 비선형 함수 F, 모듈라 덧셈, 좌측 로테이션에 기반한 16개의 동일 연산으로 처리를 진행한다.

MD5는 위와 같은 단일 연산을 64번 실행하며 16개의 연산을 그룹화한 4라운드로 묶인다. ⟨⟨⟨s는 s만큼 좌측 로테이션을 시킨다는 의미이며 ⊞은 2^32 모듈러 덧셈을 말한다.

MD5는 해쉬 함수 중 널리 사용되고 있는 함수로, MD4의 기술적 결함을 수정해 나온 알고리즘이다. 하지만 MD5에서도 보안의 결함이 발생하여 SHA 해쉬 알고리즘으로 대체되어지고 있다. 그렇지만 서명문의 인증키로 사용하지 않고 어떤 객체의 식별 코드로 사용하는데 있어서는 비교적 개발하기 쉬운 MD5를 주로 사용하고 있다.

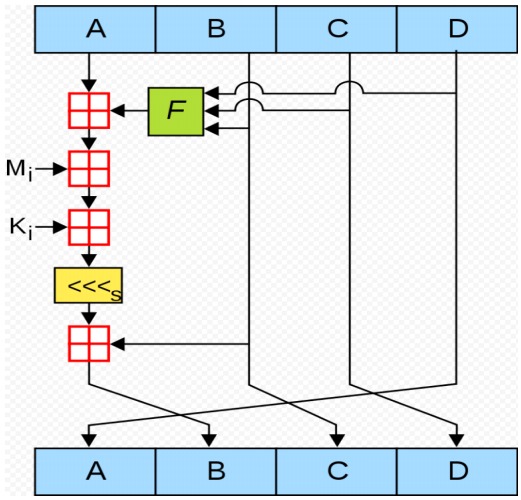

그림 1. MD5 한 라운드 개념도
Fig. 1. Block diagram of one MD5 operation

중복파일 업로드 방지 시스템에서 사용에 있어서는 보안적 기능 보다 파일의 구별 용도로 사용하기 때문에 속도가 빠른 MD5가 이상적으로 보이지만, 128비트의 비교적 짧은 해쉬 값을 생성함으로써 충돌의 가능성이 높아 적합하지 않다.

2. SHA

SHA은 MD5의 결함을 보안한 MD4기반의 알고리즘이다[4]. SHA(Secure Hash Algorithm)는 이름처럼 보안성이 강하여 보안이 중요시 되는 상황에서 사용을 권장하고 있으며, 서명문에서는 주로 SHA-2가 사용되고 있다. SHA는 MD5와 같이 512비트의 정수배로 만드는 패딩 작업을 하게 된다.

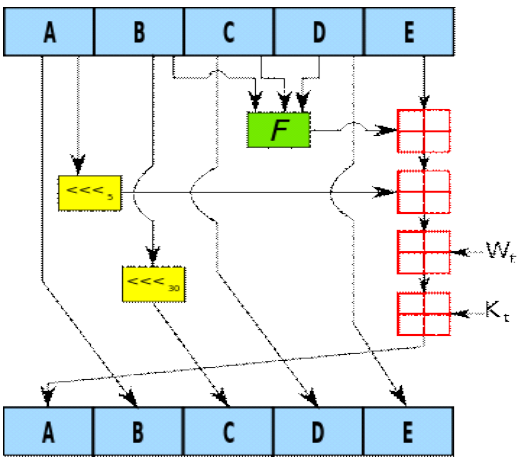

그림 2. SHA-1 한 라운드 개념도
Fig. 2. Block diagram of one SHA-1 operation

패딩된 서명문은 L*512비트로 1~512의 패딩 비트와 서명문 길이를 표시하는 64비트(K길이)를 포함하게 된다. 이 패딩된 서명문은 512비트씩 나누어져 해쉬 함수 모듈 H_{sha}에 입력된다. H_{sha}의 각 라운드의 기본 동작을 보면 입력 레지스터 A,C,D는 출력 레지스터 B,D,E에 입력되고 B는 30회 좌측 로테이션 후 C에 입력된다. 다음 입력 레지스터 E는 B,C,D를 입력으로 하는 논리 함수 f_t와 입력 레지스터 A를 5회 좌측 로테이션 시킨 값 w_t,K_t와 XOR 연산 후 출력 레지스터 A에 입력된다.

표 1. SHA의 0,1,2 버전별 특성 비교
Table 1. Comparison of SHA 0,1,2

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Word size (bits)	Rounds	Operations
SHA-0		160	160	512	$2^{64}-1$	32	80	add, and, or, xor, rotate, mod
SHA-1								
SHA-2	*SHA-256/224*	256/224	256	512	$2^{64}-1$	32	64	add, and, or, xor, shift, rotate, mod
	SHA-512/384	512/384	512	1024	$2^{128}-1$	64	80	

SHA-2(512bit)는 최대 길이가 512bit이며 충돌 경우의 수가 현존 해쉬 함수 중 가장 낮다[7]. 보안적인 측면에서는 SHA-0는 보안의 약점이 발견이 되었고, SHA-1는 방법만 제시 되었다. SHA-2는 아직 보안의 약점을 발견하지 못하였지만 기본적인 이론이 SHA-0와 동일하여 보안의 약점이 발생할 수 있다는 의견도 있다.

SHA-1은 SHA-0의 압축 함수에 비트 회전 연산을 하나 추가 한 것으로 SHA-0이 가진 암호학적 보안의 문제를 감소시키기 위해서 고안된 것이다. SHA-1과 SHA-2는 연산적 차이점은 처리 라운드 중에 Shift연산이 추가되어 해쉬 값을 생성을 하게 된다. 그리고 SHA-2에서는 알고리즘의 방식은 동일하나 해쉬 값의 길이로 스팩을 구별 하여 사용한다.

본 논문에서 제안하는 시스템 구성에 있어서는 보안의 문제점은 중요하지 않으나 출력 해쉬 값의 길이는 중요하다. SHA-2와 같은 경우에는 해쉬 값의 길이가 길어 적합하지만 연산이 많아 처리속도가 다른 해쉬 함수 보다 현저히 느리기 때문에 대용량 파일의 해쉬 값을 얻고자 하는 시스템에서는 적합하지 않다.

3. 기존 서버에서의 파일 저장방식

기존 시스템의 파일 저장 설계 방식은 다양하다. 서버 설계자에 따라 혹은 사용 용도에 따라 설계방법이 매우 다양하여 획일화된 설계방법은 없다. 서버에서 일반적인 파일 저장 방식은 시스템의 용도에 따라 디렉터리별로 분류하여 저장하는 방식이다. 분류의 기준은 날짜, 시간, 사용자 아이디, 임의의 수 등으로 구분하여 디렉터리를 생성하여 저장하게 된다. 이를 디렉터리 계층 구조라 하고, 가장 널리 사용되어지고 있는 방식이다. 구분하는 변수는 다르겠지만 보통은 이런 디렉터리 계층 구조로 파일을 보관 및 관리한다. 이런 시스템은 중복을 허용하고 파일의 해쉬 코드를 관리하지 않기 때문에, 차후 파일에 관한 문제가 발생하였을 때, 일일이 그 파일들을 찾아 삭제함으로써 관리상 문제점이 발생하게 된다. 그러나 시스템 설계가 간편하고 서로 다른 파일에 충돌 및 간섭이 생기지 않기 때문에 디렉터리 계층을 많이 사용하고 있다.

III. SA 해쉬 알고리즘을 이용한 서버 시스템 설계

자체 고안한 SA(Simple Arithmetic)해쉬 알고리즘을 이용하여 동일 파일 업로드 방지 및 저작권 관리 편의 시스템 설계를 본 논문에서 제안하고자 한다. SA 해쉬 알고리즘에 관한 소개와 시스템 설계에 관해 소개한다.

최근 인터넷 환경이 좋아지고 모바일 통신 환경까지 좋아지며 대용량 데이터들이 자주 사용되고 있다. 웹하드 서비스나 개인 클라우드 서비스를 많이 사용 되고 있다. 하지만 웹하드, 클라우드 데이터 스토리지 서비스의 서버 중에는 원천 데이터가 아닌 사본 데이터들이 스토리지에 주를 차지하고 있다. 그러한 데이터들 중 대표적으로 대용량과 중복 비율이 큰 데이터가 동영상 데이터이다. 그래서 동영상 데이터들을 많이 저장하는 서버 시스템을 설계 하는 것에 있어 본 논문에서 제안하는 설계 방식을 사용한다면 효율적인 서버 시스템 설계가 가능하다.

1. SA Hash Algorithm

본 논문에서 사용하고 제안하는 SA(Simple Arithmetic) 해쉬 알고리즘은 위에서 설명했던 MD5, SHA 알고리즘보다 중복파일 업로드 방지 시스템 설계에 있어서 빠르고 (267Mib/sec) 해쉬 값의 길이도 512bit로 길기 때문에 가장 최적화 된 해쉬 알고리즘이다. MD5의 해쉬 값 길이는 128bit인데 비해 SA 해쉬 알고리즘은 512bit인 해쉬 값 길이를 가져 충돌 가능성을 낮출 수 있다. 기존 해쉬 중 SHA-2와 SA 해쉬 알고리즘은 동일한 해쉬 값 길이(512bit)를 가지지만, SA 해쉬 알고리즘은 가변적인 길이로 설정할 수 있어 큰 시스템에서의 활용도가 SHA-2에 비해 뛰어나다. MD5는 SHA 알고리즘 보다 월등히 뛰어난 속도를 자랑하지만, SA 해쉬 알고리즘은 그러한 MD5보다 3배 이상의 처리 속도(267Mib/sec)를 가진다.

SA 해쉬 알고리즘은 현존 최고의 처리 속도를 가지며 해쉬 값의 길이에 있어서도 무한정 늘릴 수 있는 것이 SA 해쉬 알고리즘의 가장 큰 특징이자 장점이다.

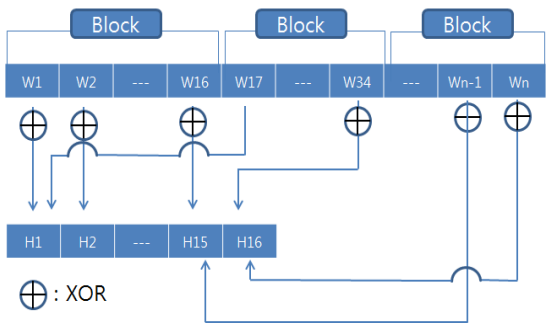

그림 3. SA Hash algorithm 구성도
Fig. 3. Block diagram of SA Hash algorithm

SA 해쉬 알고리즘은 생성될 해쉬 값의 길이(기본 512bit)를 한 블록으로 가지는 H에 메시지의 내용을 순차 적으로 XOR 연산을 함으로써 뛰어난 처리 속도를 가질 수 있다. 처리 메시지의 길이가 N이라면 시간복잡도는 N이 된다. SA 해쉬 알고리즘은 다른 해쉬 알고리즘과 달리 별도의 패딩 처리를 하지 않고 메시지가 끝나는 지점에서 연산이 끝이 난다.

```
for i from 0 to MessageLength-1
h(i mod 512) := h(i mod 512) xor m(i)
```

그림 4. SA 해쉬 알고리즘 메인 루프의 의사코드
Fig. 4. Pseudo-code of SA Hash algorithm main loop

기존 다른 해쉬 함수들과 슈도코드만 비교했을 때 main loop에서의 연산수가 MD5는 대입 연산 수가 5이고 SHA알고리즘은 대입 연산 수가 7이다. 하지만 SA Hash 알고리즘은 1로 확연한 차이가 난다.

표 2. 각 Hash algorithm의 특성과 성능
Table 2. Characteristic and performance of Hash algorithms

	해쉬 값 길이	충돌 경우의 수	속도
MD5	128bit	10^{-10}	88Mib/sec
SHA1	160bit	10^{-17}	52Mib/sec
SHA2	512bit	10^{-77}	38Mib/sec
SA	512bit (확장가능)	10^{-77} 이하	267Mib/sec

제안하는 시스템의 해쉬 알고리즘은 서명문에 사용되는 것이 아니고 대용량 파일의 특징을 표현하는 함수이기 때문에 속도의 성능이 매우 중요하다. 그러므로 제안하는 시스템을 구현하기에 있어서 기존 타 해쉬 함수를 사용하기 보다는 제안하는 SA Hash algorithm을 사용하는 것이 효율적이다.

1.1 Hash Algorithm 속도 비교

각 해쉬 함수의 메인 루프에 연산자의 개수를 표시한 표3를 보면 덧셈 연산자와 논리 연산자 그리고 Shift연산자로 구분하여 나타내고 있다[7]. MD5는 논리 연사자 보다 덧셈 연산자를 더 많이 사용되고 전체적인 연산자 수가 SHA-1,SHA-2보다 작음으로 더 빠른 속도를 보이고 있다. 그리고 SA 해쉬 알고리즘은 덧셈 연산자와 Shift연산자를 사용하지 않고 논리 연산자 중 XOR 연산만 사용하고 총 연산자의 수가 다른 해쉬 알고리즘 보다 작다. 그리하여 다른 알고리즘보다 SA 해쉬 알고리즘이 속도가 더 빠르다고 할 수 있다.

표 3. Hash Algorithm 연산 수 비교
Table 3. Comparison of Hash algorithm's operation number

Operation	MD5	SHA-1	SHA-2	SA
Addition	14	12	20	0
Bitwise operation ($\hat{}$,∧,∨,¯)	13	18	27	16
Shift operation	1	7	23	0
Total number of operations	28	37	70	16

2. Retain Count

중복파일 업로드방지 시스템 설계에 있어 하나의 파일을 다수의 클라이언트가 사용하고, 편집 권한이 있기 때문에 편집 권한과 파일의 보존 유무를 관리하는 장치가 필요하다. 이를 관리하는 장치가 Retain Count이다. Retain Count는 몇 명의 클라이언트가 편집 권한이 있는지를 Count하여 관리함으로써 편집 권한을 가진 클라이언트가 한명도 없게 되면 파일을 소멸 시키게 된다. Retain Count가 있음으로써 또 다른 장점은 서버에 있는 파일이 비정상적, 저작권문제 및 불법파일일 경우 일괄적으로 삭제나 수정함으로써 서버의 운영 및 관리의 편의를 도모할 수 있다.

Retain Count는 DB로 보관하며 DB 구조는 아래와 같다.

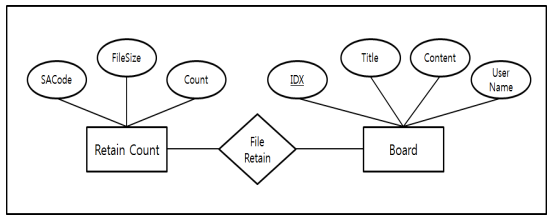

그림 5. Retain Count의 DB구조
Fig. 5. Architecture of Retain Count's DB

위 DB구조는 게시판을 예를 들어 작성한 구조이고, 파일 관리에 필요한 정보는 Retain Count 테이블 정보를 사용하여 파일을 효율적으로 관리할 수 있다.

Retain Count 테이블에서는 기본키가 존재하지 않는다. SA 알고리즘의 파일 해쉬 코드 값의 충돌 경우의 수가 낮더라도 유일성은 보장하지 못하기 때문이다. 혹 SA 해쉬 코드가 같은 파일이 있더라도 FileSize를 비교함으로써 충돌 경우 발생 확률을 낮게 할 수 있다.

시스템 설계에서 Retain Count는 클라이언트가 파일을 업로드 하기 전 SA 해쉬 코드를 전송하여 Retain Count에서 같은 SA 해쉬 코드가 있는지 확인하고 있다면 전송하지 않고 서버에 있는 기존 파일로 대체한다. 사용하는 클라이언트 수가 증가하면 Retain Count의 Count를 증가 시킨다. 클라이언트가 업로드 하고자 하는 파일이 서버에 없을 경우 Retain Count에 레코드를 생성하고 파일의 정보를 입력한다.

3. SA 해쉬 알고리즘을 이용한 서버 시스템 구성

제안 하고자 하는 SA 해쉬 함수를 이용한 중복파일 방지 시스템은 Client의 업로드 요청된 파일의 SA hash algorithm 해쉬 코드를 Meta Data Server에 전송하고, 현재 서버에서 동일한 파일이 있는지 검색하여 존재 유무에 대한 Response를 한다. 동일한 파일이 존재한다면, 업로드를

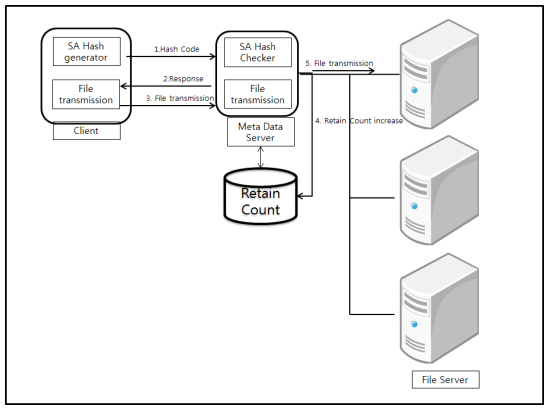

그림 6. 제안 시스템 구성도
Fig. 6. Block diagram of proposed system

받지 않고 Retain Count를 증가시킨 뒤 클라이언트가 파일 사용을 할 수 있는 권한을 부여한다. Meta Data Server에 동일 파일의 기록이 없다면, 파일을 업로드받고 Retain Count를 증가한다. 이러한 서버 시스템 설계로 업로드에 들어가는 자원인 시간 및 트래픽을 감소시킬 수 있고, 파일서버의 스토리지를 절약할 수 있게 된다.

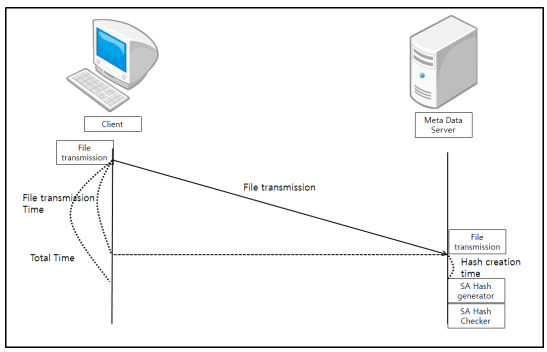

그림 7. 잘못된 해쉬 코드 생성 과정(Server Side)
Fig. 7. Procedure of making incorrect Hash code

이 시스템을 구성하는데 있어 Hash Code를 생성하는 처리는 client에서 하게 된다. 즉 Client Side가 가능한 언어로 처리를 하여야 한다. 그렇지 않고 Server Side로 처리를 하게 되면 일단 파일을 업로드 받고 나서 추후 과정들을 이어나가기 때문에, 기존 서버 시스템 설계보다 비효율적인 설계가 될 것이다. 제안 시스템을 구성하는데 있어 가장 주의해야 되는 부분은 Server Side와 Client Side이다. Client Side로 시스템을 구성하여야 업로드 이전에 판단 및 처리하여 효율적인 시스템 구성과 본 제안 시스템의 성능을 올바르게 활용할 수 있다. 하지만 기존에 있는 Client Side Script언어는 client 로컬 파일을 읽는 것이 보안에 위배되어 사용할 수 없었다. client 로컬 파일을 읽기 위해서는 웹 표준이 아닌 Active X와 같은 비표준을 사용하여야 했다. 그러나 최근 HTML5가 개발됨으로써 HTML5가 client 로컬 파일을 보안 위배 없이 웹 표준으로 사용이 가능하게 되었다. 그래서 본 시스템의 구현이 다른 제약 조건 없이 가능해진 것이다.

Client Side로 해쉬 코드를 생성을 하고 서버에 해쉬 코드만 전송하게 된다면 실험 결과 1Gibibyte 파일을 전송하는데 평균 14.57초가 걸리고 기존 일반 전체 파일을 업로드 할 때에는 평균 139.27초가 걸리게 된다. 평균적으로 89.6%의 시간 절약 효과를 나타낸다. 하지만 일치하는 해쉬 코드가 없을 경우에는 해쉬 코드 생성 시간과 파일 전송 시간이 걸리기 때문에 해쉬 코드를 생성하지 않고 바로 전송하는 일반 시스템 보다 약간의 지체 시간이 걸리게 된다. 그래서 서버 파일 중 중복률이 많이 발생하는 서버 시스템 설계에서 특히 효율이 높게 나타난다.

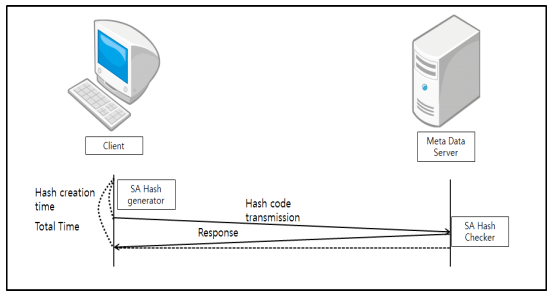

그림 8. 올바른 해쉬 코드 생성 과정(Client Side)
Fig. 8. Procedure of making correct Hash code

본 논문에서 제안하는 시스템은 파일의 누적과 중복률이 증가할수록 효과 적으로 시스템 구성을 할 수 있고 중복률이 11%이상이 될 경우에 효율적이라 할 수 있다. 만약 0~11%의 중복률을 가지는 시스템에서 본 논문에서 제안하는 방법으로 시스템을 설계를 하게 된다면 해쉬코드 생성 전송, 동일 해쉬 코드 존재 여부 확인, 그리고 Response까지의 시간 그리고 파일전송 시간이 걸리기 때문에 더 많은 시간이 필요해 비효율적이게 된다. 하지만 중복률이 11%이상이 될 경우 효율적인 시스템 구성이 가능하고, 중복률이 55%이상이 되면 시간적 감소는 물론, 스토리지 공간이 50%이상 감소할 수 있게 된다.

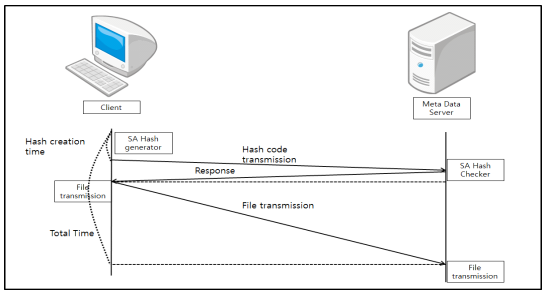

그림 9. 일치하는 해쉬 코드가 없을 경우
Fig. 9. Action in case there is no same hash code

Ⅳ. 성능 분석

해쉬 알고리즘을 웹 환경이 아닌 일반 프로그램으로 실행하였을 때 속도 비교표이다. 시간 측정은 해쉬 코드 생성할 파일을 RAM에 로드한 후 순수 해쉬코드를 생성하였을 때의 시간을 측정하였다. 다른 변수들을 제거하기 위해 신뢰도가 높은 RAM에 로드한 다음 진행을 하여 RAM과 CPU간의 연산으로 처리 시간만을 측정하여 다른 변수들이 발생할 수 없게 되는 것이다. 실험은 동일 파일로 50회의 실험 결과의 평균을 산출한 것이고, 각 실험에 있어 근소한 시간적 차이는 보이지만 평균치에 근접하였다. 동일 알고리즘에 동일 파일의 최대값과 최소값은 0.2초 이내로 편차가 났다. 본 논문에서 제안하는 SA알고리즘은 초당 267Mib처리로 가장 빨랐으며 기존 해쉬 알고리즘 중 가장 빠른 MD5보다 3배 이상의 속도를 보이고 있다. MD5는 초당 88Mib의 속도를 나타내었으며 알고리즘의 복잡도가 높거나 해쉬 코드 길이가 길어질수록 처리 속도가 감소하는 것을 볼 수 있었다.

표 4. 실험 환경
Table 4. Environment of experiment

S/W 스펙	Client	운영체제	Windows 7
		개발 환경	Visual Studio 2005
	Server	운영체제	Cent OS 5.5
		개발 환경	APM(Apache,PHP,Mysql)
H/W 스펙	CPU	Intel Core2 Duo E8400 3.00Ghz	
	RAM	4G	

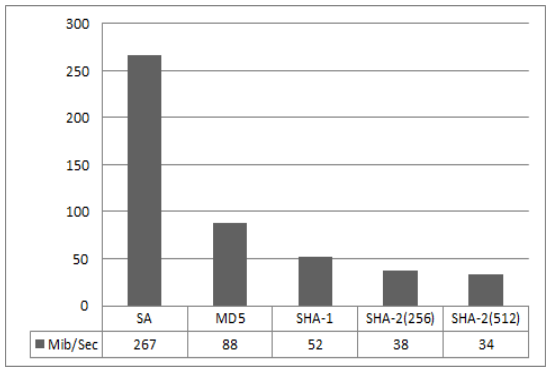

그림 10. Hash algorithm 속도 비교
Fig. 10. Comparison of Hash algorithm's speed

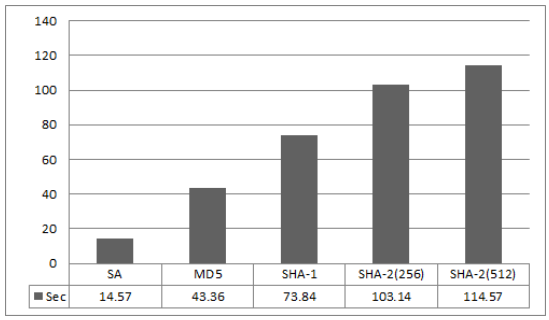

그림 11. 웹 상의 해쉬 코드 생성과 Response까지의 시간
Fig. 11. Response time on Web

위 그림은 1Gibibyte 파일의 해쉬값을 Client Side로 생성하고 서버에 전송 후 동일한 해쉬 코드가 있는지 검사하여 클라이언트에 Response를 해주는 프로세서까지의 시간을 측정한 자료이다. HDD에서 파일을 RAM에 로드하고, Client Side의 웹 환경에서 해쉬 코드를 발생하여 생성한 해쉬 코드를 서버에 전송한 뒤 동일 해쉬 코드가 존재하는지 Response를 해주는 시간을 측정한 실험이라 앞서 했던 해쉬 함수의 연산 속도 비교 실험보다는 시간이 더 많이 걸리게 된다. 하지만 본 시스템의 구성에서의 모든 프로세서가 포함이 된 가장 정확한 속도 비교의 자료이기도 하다. 실험은 동일 파일로 각각 50회의 실험한 평균치를 나타내었고, 최고치와 최저치의 편차는 2.35초 이내로 평균치에 크게 멀어지는 실험값은 나오지 않았다. 1Gibibyte를 처리할 때 SA 해쉬 알고리즘은 가장 적은 시간을 소모한 평균 14.57초로 나왔고 MD5는 평균 43.36초로 나왔다. 웹 환경에서도 SA 해쉬 알고리즘은 가장 빠른 처리 속도를 보이고 있다. SA알고리즘과 해쉬 코드 길이가 512bit로 같은 SHA-2(512)와 비교 하였을 때도 12%의 시간 소비만으로 연산을 처리하는 것을 볼 수 있다.

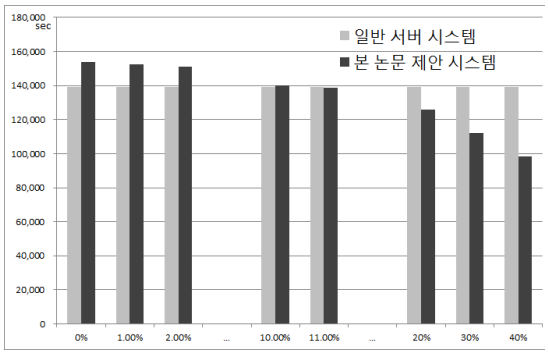

그림 12. 일반 서버 시스템과 제안한 시스템 성능분석
Fig. 12. Performance analysis of proposed system

위의 도표는 일반 전송 방식인 중복을 허용하는 시스템과 본 논문에서 제안하는 SA 해쉬 알고리즘을 이용한 시스템의 성능을 분석한 표이다. 실험은 1Gibi-byte 파일을 1000개 중 중복률을 parameter로 한 실험을 진행하였다. 설계 방법 선택에 있어 이 성능 분석 자료가 가장 중요한 자료가 될 것이다. 중복률에 따라 일반 서버 시스템과 본 논문에서 제안하는 시스템의 성능에 차이를 보이고 있고, 설계를 하고자 하는 시스템의 중복률이 선택의 기준이 된다. 실험에서의 업로드 속도는 평균 7.18MiB/Sec이다. 업로드 속도가 빠를수록 일반 서버 시스템 방식이 유리해 지며 실험한 업로드 속도는 일반 인터넷에 비해 빠른 속도의 환경에서 실험을 하였다. 중복률이 0~10%까지는 일반 서버 시스템이 효율적이나 중복률이 11%이상이 되면서 본 논문에서 제안하는 SA해쉬 알고리즘을 이용한 서버 시스템이 효율적인 것을 확인할 수 있다.

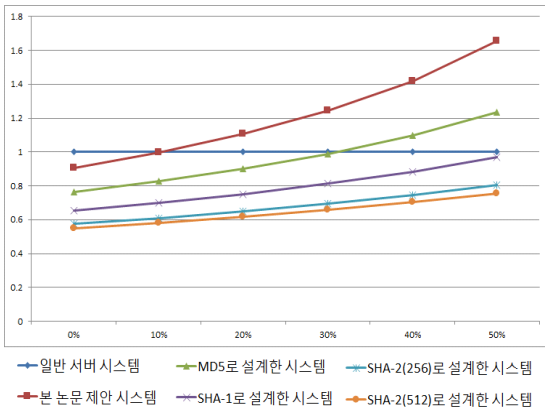

그림 13. 각 해쉬 함수로 설계한 시스템 성능 비교
Fig. 13. Performance comparison of systems using Hash code

해쉬를 이용한 서버 시스템 구성에 있어서 어떤 해쉬 함수 알고리즘을 사용하여 설계하였나에 따라서도 성능의 차이가 발생한다. 앞서 일반 서버 시스템과 SA 해쉬 알고리즘을 사용한 서버의 성능은 비교하였다. SA 해쉬 알고리즘이 아닌 다른 해쉬 알고리즘으로 설계하였을 때의 성능 분석이다. 기존 해쉬 알고리즘 중 가장 빠른 MD5가 중복률이 30%이상일 때 일반 서버 시스템 보다 효율적으로 나타났고 그 이 외의 해쉬 알고리즘인 SHA 계열의 해쉬 알고리즘은 중복률이 50%가 되어도 일반 중복을 허용한 시스템 보다 비효율적으로 나타났다. 스토리지 절약에는 어떤 해쉬 알고리즘을 사용해도 되지만 처리 속도가 너무 증가하게 되면 시스템의 효용성이 없게 된다. SHA 해쉬 알고리즘은 이렇게 대용량 데이터의 해쉬 값을 생성하는 것에 사용하고자 설계된 알고리즘이 아닌 보안의 서명의 해쉬 코드를 발생하고자 개발된 알고리즘이라 속도적인 면에서는 느린 것이다. 이 처럼 중복률이 11% 이상인 서버 시스템 설계에 있어 SA 해쉬 알고리즘을 이용한 서버 시스템 설계가 효과적이다.

V. 결 론

일반 서버 시스템에서는 클라이언트가 업로드를 요청할 경우, 서버에 동일 파일의 존재 유무와 상관없이 업로드를 받아 저장하게 된다. 그렇다면 서버의 스토리지 공간에 중복되는 파일에 의해 필요 없이 공간을 점유하게 되고, 업로드 시에 트래픽과 시간 소요가 발생하게 된다. 일반 서버 시스템 설계에서의 이점은 설계의 편리성과 하나의 파일은 한명의 클라이언트만이 편집 권한을 가져 권한 설정에 있어 관리가 편리하다. 본 논문에서 제안하는 시스템은 클라이언트가 업로드를 요청할 때 파일의 해쉬 코드를 서버에 전송하여 서버에 동일한 파일이 있는지 해쉬 코드로 검사한다. 동일한 파일이 존재한다면, 업로드를 받지 않고도 기존파일의 사용권한을 얻을 수 있다.

업로드 중복방지 시스템에 최적화된 SA 해쉬 알고리즘은 기존 해쉬 알고리즘 중 가장 빠른 MD5보다 3배 빠른 속도를 가지고 있으며, 해쉬 값의 충돌 여부를 결정하는 해쉬 값의 길이가 512bit로 SHA-2(512bit)와 동일하고, 필요에 있어 그 이상의 길이로 설정이 가능하다. 성능 분석에 있어서는 기존 해쉬 함수들 MD5, SHA-1, SHA-2와 SA 해쉬 함수의 속도를 비교함으로써 제안하는 시스템의 가장 적합한 해쉬 함수가 SA 해쉬 함수인 것을 증명하였다. 기존 해쉬 알고리즘을 이용해 중복파일 업로드를 방지하는 시스템에서는 기존 해쉬 알고리즘 중 가장 빠른 MD5가 중복률이 30%이상일 경

우 일반 서버 시스템 보다 효율적으로 나타났고 SHA-1, SHA-2는 중복률이 50%가 되어도 일반 서버 시스템 보다 느려 해쉬 값으로 중복 파일 업로드 방지 시스템 설계에 있어 사용하는 것은 비효율적인 것을 알 수 있다. 일반 서버 시스템과 SA 해쉬 알고리즘을 사용한 서버 시스템 성능 분석으로 파일의 중복률에 따른 데이터 처리 시간을 측정을 하여 중복률이 11% 이상이 되는 경우에 SA 해쉬 알고리즘을 사용한 서버 시스템 설계가 더 효율적인 것을 알 수 있다. 그래서 본 논문에서 제안하는 SA 해쉬 알고리즘을 이용한 서버 시스템 설계는 일반 웹 게시판과 같이 중복률이 낮고 파일 용량이 작은 시스템 보다 중복률이 높고 파일의 용량이 큰 시스템에서 효용 가치가 높다고 할 수 있다.

향후 과제로서는 해쉬 값의 충돌을 방지하는 부분이 본 시스템 설계에 있어 가장 위협적인 문제점이기에 충돌을 방지할 수 있는 방법 연구와 하나의 파일을 여러 클라이언트에서 사용 권한을 부여하고 있기 때문에 사용 권한 관리 방법이 추가적으로 연구되어야 한다.

참고문헌

[1] Berson, Thomas A.. "Differential Cryptanalysis Mod 232 with Applications to MD5". EUROCRYPT. pp. 71-80, Nov 1992.

[2] Bert den Boer, and Antoon Bosselaers, "Collisions for the Compression Function of MD5", Berlin London, Springer, pp. 293-304, 1993.

[3] Hans Dobbertin, "Cryptanalysis of MD5 compress.", Announcement on Internet, *"http://Citeseer.ist.psu.edu"*, May 1996.

[4] Dobbertin, Hans, "The Status of MD5 After a Recent Attack", CryptoBytes, Vol.3, No.2, pp. 9-14, 1996.

[5] Xiaoyun Wang, and Hongbo Yu, "How to Break MD5 and Other Hash Functions", Lecture Notes in Computer Science Volume 3494, pp. 19-35, 2005.

[6] Stéphane Manuel, "Classification and Generation of Disturbance Vectors for Collision Attacks against SHA-1", Spring Science Business Media, Volume 59, Issue 1-3, pp. 247-263, April 2011.

[7] SHA-1 wikipedia *"http://www.saylor.org/site/wp-content/uploads/2012/07/SHA-1-1.pdf"*

[8] Ho-Min, Jeong, "Design of Deduplication Supported Clustering Backup System using File Finger Printing", Korea Information Processing Society, Vol 14 No 2, pp. 0737-0740, Nov 2007.

[9] Jung Hoon Kim, Byoung Hong Lim, Young Ik Eom, "Eliminating Redundant Data for Storage Efficiency on Distributed File Systems", Korea Information Processing Society, Vol 16 No 2, pp. 0111-0112, Nov 2009.

저 자 소 개

황 성 민

2009: 안동대학교
정보통신공학과 공학사.
2012: 안동대학교
정보통신공학과 공학석사
2013 - 현재: 안동대학교
정보통신공학과 박사과정
관심분야: 컴퓨터 보안, 컴퓨터네트워크
Email : iris8037@nate.com

김 석 규

1990: 연세대학교 전자공학과 공학사.
1992: 연세대학교 전자공학과 공학석사.
1997: 연세대학교 전자공학과 공학박사
1997-2004: SK텔레콤 선임연구원
2004-2006: 연세대학교
전기전자공학부 연구교수
2006 - 현재: 안동대학교
정보통신공학과 부교수
관심분야: 컴퓨터네트워크,
센서네트워크,,
차세대네트워크, 이동통신,
네트워크 보안
Email : sgkion@andong.ac.kr